Managing Trust between collaborating Companies using outsourced Role Based Access Control

Thomas Hildmann <hildmann@prz.tu-berlin.de>
Jörg Bartholdt <jorgb@hpl.hp.com>
Technical University of Berlin, Hewlett Packard Laboratories Bristol

Abstract

In this document we describe an approach for modelling large organisations applying an RBAC-schema to control access to remote services of the organisation. The model is object-oriented, non-hierarchical and divides the organisation into different contexts (posts, groups, persons, services, resources) in which roles are defined local to that context. We explicitly address the problem of access controlling the policy information itself by using the same means as for external resources. Therefore, this approach enables policy information of each context to be managed by another person, namely the one responsible for that very resource context, and makes it superior over strict hierarchical models.

The first version of our model was designed to manage the access in one defined organisation. This paper will show how this concept is expanded so that it can also be used for access control between collaborating companies.

As an example we will show how an electronic market place can be modelled and each partner in that marketplace can manage access policies to his catalogues, prices, discount rules, orders, etc. on his own while the integrity of the whole marketplace is assured.

1 Introduction

Role-based access control is a means to keep access control maintainable in large organisations and for many services. Large organisations have a strong need for intuitively managing the rights and duties of their staff as well as external partners concerning the organisation's resources. Our first iteration on extending this modelling approach initially targeted the modelling of (public) administrations[1]. It also proved to be a perfect fit on a follow-on project[2] investigating e-commerce market places.

We decided to use a role-based and object-oriented approach, aiming to express policies on an application level. Following the "divide and conquer" principle, we separate different contexts of the organisation *(posts, groups, persons, services, resources)* and define roles local to those contexts. This allows us to concentrate on a small part of the organisation that we wish to model. It also allows the possibility to distribute responsibility for managing the policies (control access to the access control information itself).

During our work on the current project we noticed several problems coming up if the access model leave the organisation boundary. But not only in trust relations between several organisations it can be important to hide relationships between roles in an RBAC system even if like in our model an entitled user can manage parts the model himself.

This document will give a short overview in section 2 of the features and drawbacks of standard role-based access control (RBAC).

Section 3 will describe an example by modelling an e-commerce marketplace and will give a summary of benefits and disadvantages of our classical single organisation model.

Section 4 will define a revised model to fix this problems and will try to support multilateral security to our RBAC system.

Section 5 discusses the outsourcing concepts which are possible with the revised model.

Section 6 and 7 are going in implementation details and architectural decisions.

---

[1] This work was carried out as part of the E2S (End-to-End security over the internet) project no. 20563, funded by the EU commission. It ended in December 1997.

[2] MultiPLECX, EU funded project no. 26810, scheduled March 98 until March 2000

Section 8 will give a conclusion and outlook to future work.

2 Role-based Access Control (RBAC)

Classic access control is based on the individual (*subject*) accessing a resource (*target*). Access control lists (ACL) are stored with the resource and access is permitted by UID/password authentication and permission checking (see Figure 1). Some other ways of storing the permission exist, but are not widely used: credentials are issued by the resource or the controller of a resource and stored at the user's side. Accessing the target, the user presents the credential which is verified by the resource and access is granted appropriate to this credential. Revocations of issued credentials must be stored with the resource and checked with each presentation of a credential.

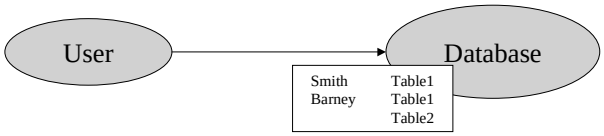

Figure 1 - Access control list based access control

Roles are placed between the user and the resource (Figure 2) and describe rights and duties people have to perform tasks [Grimm91]. Role descriptions can be found for example in post descriptions of organisations. Roles provide an *extended viewpoint* on access control. Their notion must not be confused with *means* for implementing access control such as ACLs.

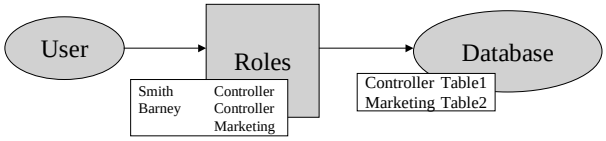

Figure 2 - Role-based access control

If the number of subjects and targets grow, individual access control becomes infeasible. Each individual needs to be assigned each access right by hand at each service location. The same applies to the revocation procedure if the person changes the post or leaves the organisation. Roles greatly simplify the administration by assigning access rights to roles and roles to subjects.

In contrast to common access controls, which are either implemented as *access control lists* (ACL) and reside at the target of the access or as *credentials* and reside at the user side, the access control information in an RBAC system resides in between.

Two roles can be marked as "mutually exclusive". This means that each role contains at least one access right which is incongruous with the other. In a bank for example, this could be the role "cashier" and "controller": A controller cannot audit his own work [Kuhn97].

In contrast to RBAC models like [Sand97] we do not use a strict hierarchy (tree-like structure). Our model is a not directed graph of relationships.

Descriptions of RBAC systems often make the assumption that the role system is more or less static. They do not address the problem of administration of the role system itself. A simple solution is to have a role administrator, who may assign and revoke roles and rights. This concentrates the power of the system on a single person. The role admin changes the model to whatever departments tell him, as the number of participants and resources grow. This makes the system vulnerable to false orders. Responsibility should be left to those who are quite familiar with the context of a resource.

Ravi Sandhu et al. have created a model ARBAC'97 dealing with the administration of an RBAC system [Sand97]. The separation of this part gives a hint to the importance of this topic.

The above example of a cashier and a controller changes if the person is cashier at a certain branch, but controller at another one. The exclusiveness can be interpreted differently, if the roles are seen in their context. Luigi Giuri and Pietro Iglio propose the use of role templates in their article [Guir97] to address this topic. Roshan Thomas [Thom97] looks at it from another perspective. He firstly structures the contexts (called *Teams*) and then applies roles to them. Our model is similar to this. But as you will see later we are also using some kind of templates to describe a business to business relationship. The idea is to define the interface between two organisations on a role level (e.g. who is the person to contact, who is the seller, etc.). We called this kind of templates interfaces. These interfaces are used like interfaces in Java to define the look-like of classes (e.g. methods which have to exist and its parameters). Our idea of interfaces covers little bit more as discussed in section 4.

3 A sample modelling

To figure out the difficulties which come up when running an RBAC system in a multilateral security environment we will model a marketplace as an example.

We will start to give an example for a single organisation and than expand it to a multiparty model. Searching for an appropriate model to describe the rights and duties of the staff/members of the organisation, we first need to describe the organisation itself and the relations between its entities. The access rights will be expressed on an *application level* (we will explain later why). The set of granted access rights are part of the roles that people play in the organisation. Each staff member, for example, is assigned to some roles (most of them are because of his/her post, others because of a current working context). We aim to build a *uniform system* for managing policies for accessed resources as well as for the policy model itself.

A thesis of this work concludes that an access control system for a large community can only work, if access control information is maintained by those people who are familiar with the content and context of the controlled information. The distribution of the responsibility is in our view a promising way to manage such a system efficiently.

Our model uses an object-oriented approach to model the enterprise structures. We first try to identify the objects in our scenario which are the resources (i.e. a database) and contexts (i.e. a department or workgroup). We define the term *resource context* for these objects. *Roles* and *access rights* will apply only to a particular object and do not interfere with those of other objects.

Relationships can be modelled by structuring the resource contexts and their roles using *memberships*. We do not demand a role hierarchy. Memberships can be from any role to any other role which leads to a web of memberships (compare to Figure 6).

The usage of memberships instead of a hierarchical structure is used to support the workgroup idea instead of the idea of a company hierarchy. It is at least a better idea if a business to business relationship has to be modelled. Each business partner wants to be with equal rights. Even in the model.

3.1 Resource contexts

The marketplace will consist of a set of web-based catalogues including the order and payment/billing process. Prices in the catalogue may differ from customer to customer according to special mutual arrangements (discounts, exclusive sellers, etc.) between a supplier and customer. The policy model shall describe the rights (i.e. the right to get a certain percentage discount for a certain category of goods from one supplier, or the right to update a supplier's catalogue) of all the participants involved in the marketplace.

In a virtual marketplace, we find several *users*, *buyers* and *sellers*, as well as *broker*, *design agencies* (for a web catalogue, etc.) and the organisation which actually owns the marketplace. Figure 3 shows a single organisation and its internal contexts.

With the separation of entities, we clearly identify the scope of each company/organisation.

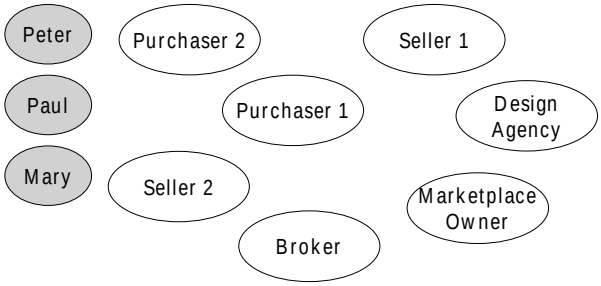

Figure 3 - separation of resource contexts

3.2 Classes of resource contexts

The next step is to identify possible *classes*. Classes identify a typical type of resource context (i.e. a Buyer class). Those classes define the role- and access right-structure. So, instantiating an object of that class is more convenient and less error-prone than creating an empty resource context and set up all roles and access rights on your own every time. So, those classes serve as some kind of template.

At the end of this section we will pick up the topic classes again.

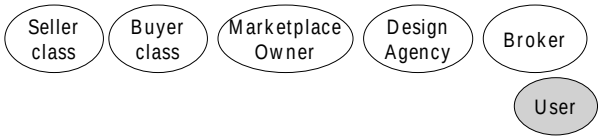

Figure 4 - Classes of resource contexts

Figure 4 contains the vital classes of an e-market place. There is a purchaser and seller class, as well as a broker class and a design agency class. There might be only one instance of the marketplace class. Note, that each class depicts a certain context in a marketplace.

3.3 Applying roles

After successfully separating the contexts, the next task is to identify a typical set of roles for each class. On the seller's side, we will have a catalogue updater, a price updater, an administrator, somebody who collects the orders, etc. On the purchaser's side, we will have a buyer, a sanctioner, a sourcer, etc. (Figure 5).

Special arrangements between a customer and a supplier are also modelled. On the supplier's side we introduce discount roles, on the purchaser's side a preferred supplier role. For example, the Sourcer can identify sellers as preferred sellers, which means either that a buyer of this Purchaser organisation may only buy from that seller, or at least, that the items of the seller are displayed on the top of the page.

When the catalogue is built, the process can determine whether the current purchaser gets a discount on the queried product by asking the model about the purchaser's membership of the discount role.

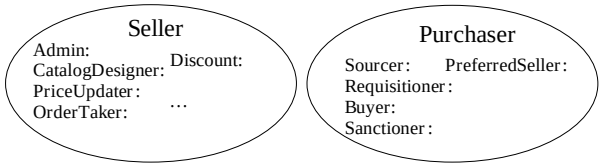

Figure 5 - Roles of a e-market place

3.4 Memberships

Now, we can have a look at Figure 6, showing some instances of those classes in a marketplace.

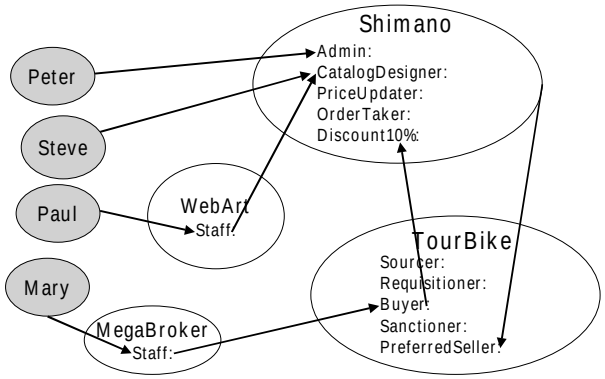

Figure 6 - Memberships

There is a selling organisation "Shimano" which produces parts for bicycles, and a purchasing organisation "TourBike" which builds complete bikes.

Peter is the Admin of Shimano, Steve is the CatalogueDesigner. Now, think how Shimano could outsource the catalogue design to an art company. Paul is designer at WebArt, which Shimano just adds as a member to its CatalogDesigner role, and thus providing Paul with all the rights of a CatalgueDesigner at Shimano.

The same can be true for the Purchasing organisation, which outsources the buying process to a broker.

TourBike and Shimano agree on a contract that gives TourBike a 10% discount on all items, Shimano sells and TourBike commits to buy those parts only at Shimano. This bilateral agreement is modelled like this: Shimano adds the buyers of TourBike to the discount role, while TourBike (in this case, somebody in the role of a Sanctioner) adds Shimano to the list of preferred sellers.

3.5 Benefits

All the marketplace processes (displaying catalogue pages, creating bills, etc.) can rely on gathering the necessary information from the single source of trust: the Model.

This gives a lot of advantages over other role-based approaches:

The Model is not only capable of performing access control for other applications, but it also controls access to its management functions itself.

The Model has no need for a single role administrator. Each entity (in the marketplace model, each Company) has somebody in charge of maintaining the proper access rights and roles memberships.

Changes made in one entity do not influence the other entity. The ability of change I.e. the member list of a role in TourBike, does not enable you to change any role elsewhere.

Administrators only deal with a context they understand. The admin of Shimano is supposed to know what a role is good for and who (respectively, what other roles) are members.

The model does not force you to have classes of objects (in fact, you may have lots of objects which are completely different from each other), but it helps understanding the "big picture" and simplifies administration. By building classes of entities, you have the possibility to predefine certain roles and access rights (like templates). This makes it feasible for non-computer experts to use such a system. Roles can be deleted or new roles can be created in an certain instance later. If, for example, a Purchaser organisation does not need approval of orders, it can delete the Sanctioner role. It is up to you, when you install this system, whether you will allow objects to change their set of roles in your particular environment. In the marketplace environment, we restrict the model by not allowing changing of the role set of an object.

3.6 Disadvantages of this model

The problems on modelling the marketplace this way are:

- The marketplace provider is able to see, change or delete any information of the model even those which do not belong to his company because the marketplace has the system and the storage systems the model is running on.
- Each company is able to see the role membership of any other company. So it is possible to see special contracts like the “preferred Seller” contract. This can not be acceptable for companies using such a model. Only the partners of a contract have to be able to see the modelling of this contract.
- We do need a very fine granularity of the visibility of relationships. It must be possible to understand the memberships between the organisation internal objects. But this has to be hidden for external viewers. To give an example: The important question for the marketplace is only: Has this user the access right to this resource and not how (s)he granted to have these access rights?

The last point was solved with a modelling of internal and external representations of a company with different access rights. But this was not what we wanted to model.

4 The revised model

The solution for these disadvantages is to separate the model and run it on a distributed system called the TrustManager. Each TrustManager instance holds the information for the own company only. To make decisions about the access right of a user it must be possible to forward the query to TrustManagers of other companies. This makes it possible to hide information of the internal structure of the company and only answer requests which are necessary for the co-operative work between the companies.

Our initial model was designed to depict the structure of a single organisation. Managing several organisations also needs other classes of entities and different behaviours. So the new model is based on a meta-model which first defines the existing classes in this model, its behaviour and access rights. This is realised with a strong object-oriented definition language[3].

The expansion to this concept is a condition part in each method which have to be guaranteed for execution. To simplify the modelling we also provide a condition overloading of methods. So it is possible to define several implementations of a method with same parameters and return-values but different conditions. The system checks every condition and executes the first implementation matching the condition. A condition can be used for example to divide internal access from external usage of the TrustManager.

An example: The same method can be used to identify a buyer of a given organisation from outside this organisation. But in this case no more information is given or just a brief comment with an e-mail address for the organisation’s representative. Calling this method from inside the organisation also returns the department. Calling it from inside the department it returns the initiator of a given transaction.

To provide and use services from and to other companies a well defined interface must be worked out. This interface has to be interpreted by applications and by users.

To have a look on our marketplace again: The catalogue software has to know which methods of which object can be called on the TrustManager of a given organisation. The object can be found in a member list of a role “buyer” or “seller” on the marketplace itself. The method which has to be called and the semantics behind the answer has to be defined in the interface.

An interface from our point of view has always two parts. A technical part (the interface definition written in RBAC model

[3] This is described in an internal paper of Stefan Lenz: “Transaction based distributed modelling with extended RBAC systems”, MultiPLECX / TUB, 1999

notation) and a contract part (written in human language). Note that the model itself is not obligatory.

On the other hand for each contract that two companies make they have to define an interface for there RBAC models.

The implementation of an interface can be a condition for the execution of a method. So it is possible for the provider of a service to check the technical points of the contract at each invocation of the method. The role membership ends with a break with a given interface.

Another important advantage is the hiding of method internals. An example: The marketplace software is able figure out if a user gets discount from a given company. But it can not get the reason. The mechanism behind at the seller company can be very complex (bonus lists, special prices, contracts, special days, time, etc.).

5 Outsourcing concepts

There are two points concerning outsourcing concepts: It is possible to source out the RBAC model to a trusted third party and it is possible to make a classical outsourcing of duties and the access rights to fulfil these duties.

5.1 Outsourcing of RBAC models

The concept described in the previous section is the most secure one. But not every company has its own internet leased-line connection, no qualified persons to manage an RBAC system or even not the manpower to have their own TrustManagers running.

In these cases it is possible to outsource the models to other existing TrustManagers in trusted third party environments and manage the role membership association over internet. An other possibility is to source out all RBAC related work to third party services.

This last possibility is very important for our marketplace provider who wants to acquire non-business users. These users do not want to manage any access rights or even care about it. They are mostly single persons who just want to use the catalogue and make orders.

5.2 Outsourcing of access rights

Another concept of outsourcing which we are support with our model is the outsourcing of rights and duties to other companies.

Coming back to the example of the web-designer company WebArt and Schimano which produces parts for bicycles, Schimano has got the access right to its catalogue pages and forward this access right to WebArt. This is the classical example of outsourcing services in business to business relationships.

As you remember: It can be modelled that it is not possible for an external viewer to figure out why a user has the right to access. So the arrangement of outsourcing is hidden.

6 Architecture

Before going to the details of the model, we will describe the architectural requirements and assumptions about the environment in which such a system can be used.

The idea is, that there are several applications (resources) which rely on some kind of policy decisions (mainly access control decisions).

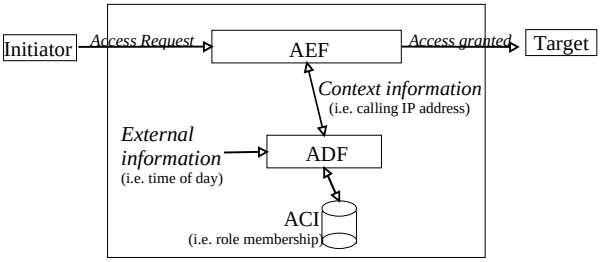

Figure 7 - Access Control Enforcement (AEF) and Decision (ADF) according to X.821

Applications shall enforce policy control at each of their entry points (i.e. server processes listening on a UNIX port, CORBA servers waiting for incoming calls). The application shall decide what kind of operation is requested and what access rights are necessary to perform the operation. It then asks the Model Interpreter and provides the context information about the requested operation (who, what, etc.). The Model Interpreter acts as a decision function and consults its model that provides context information and maybe some other source of information like the time of the day to decide. [X.821]

The model and the model interpreter should be built in a way that they do not have to know anything about the application they serve. On the one hand, we don't know which kind of application will come and on the other hand, adding another application to the system must not result in recompiling/implementing the Model Interpreter.

We take a similar approach as [PM]. To check the access rights of a user a method with a given name is called. This method returns true of the access is accepted. It is a question of the application to call the right methods and interpret the results.

Another assumption we make is that we want our system to be safe. Everything which is not explicitly allowed is forbidden. So, *access rights* allow one to do something.

Each application and each organisation served by the access control model can run its own policy and can implement it in different ways.

7 The Details

We have seen how the separation into resource contexts gives us the ability to deal with a large organisation because we have to think only about a small piece at a time. We can also distribute the work (and right) of maintaining all the resource contexts to different responsible persons without interfering with each other.

We will now cover the details of resource contexts, roles, memberships, etc.

7.1 Resource contexts

We use the term *resource context* both for representing *remote services* and to model the *enterprise structure*. This integration

meets the requirement to use the policy model for controlling remote information as well as the policy model itself.

A *resource context* is the computational representation of a real (i.e. persons/employees, database, legacy systems) or abstract (i.e. groups, posts) entity. The model consists of one or more *resource context*. Note, that the term *resource context* will be used in conjunction with "object" and "entity".

An *resource context* consists of

- an ID, which identifies the object. The system does not use a hierarchical naming convention. All objects share the same name space,
- in addition it is now possible, to model a “contain” of objects,
- a set of roles,
- a set of attributes, which can give information about the object to the access control system or the users. Access control decisions can depend on that information. Applications can store some state information about the entities.
- methods which can be called by clients. These methods are for managing the policy model

In fact the access to attributes and roles are methods, too but we make this distinction to emphasise the special semantic of these sets of methods.

7.2 Roles

Roles represent a task of a person within a *resource context*. This includes rights, duties and descriptions on how to achieve the aim of the role. This nature of roles meets the requirement of posts within an organisation. As duties and acting patterns are difficult to model for an electronic system (to compel a person to do something must be part of regulations of the organisation, not part of the electronic processing environment), we limited our efforts to the rights-management.

Interface and the check for an implementation of such an interface can be considered as a base for supporting duties. It is also possible to check the result of duties which can be expressed over algorithms (making backups for example).

We start from the hypothesis, that an efficient way for maintaining access control in large organisations is to *leave responsibilities* for assigning access rights to people, who are familiar with the content and context which is accessed. This also enables each *resource context*-responsible person to define certain aspects of the *related security policy* which is appropriate for that context.

Roles are declared local to each *resource context* as they are only meaningful in the appropriate context. Roles provide members of the role with access rights concerning the *resource context*, i.e. the right to read information concerning the stock data.

The role structure of each *resource context* can be defined without influencing any part of the rest of the system, especially no other *resource context* and role definition. A role is part of exactly one object and consists of

- an ID, which identifies the role. Role names are local to the *resource context*, so two *resource contexts* may have roles with the same name, but the two roles are not linked in any way. Whether an application wants to put more semantic to this, is up to the application. For example, it makes sense, that classes of objects have the same roles, so that the application can rely on the semantic of a role name like "buyer". This may vary from deployment to deployment of the model.
- a set of members, which are either users or other roles
- an optional mailbox (mailbox deals with using the model for a messaging system which is out of the scope of this document. Refer to [Nagel98][F1_E2S])

7.3 Membership

An object itself does not provide a good flexible mechanism for an access control system. By connecting the object through memberships enables the propagation of membership access rights and duties. This will build up a web of objects.

Membership is a unidirectional connection between two roles of different or the same objects. A role of an object (<O1.R1>) takes a role of another object (<O2.R2>) as a member, which means that it will give access rights to that role and all its members.

Through the chains of memberships, it’s easy to forward access rights to others, but not bother about each individual access right explicitly. It is sufficient to be member at a role which has itself a membership established to a role of the intended target, to get access. For example, members of a commission are members at publication server, which means they can write public documents to that server. If the commission asks a secretary to write those papers and publish them directly, they can do so by adding the secretary as a member of the commission. They do not have to inform the publication server administrator to set up access for the secretary.

There is no limit to the number of indirections.

There is no explicit root element and no strict hierarchy. However the model is capable of building tree hierarchies with one root element, if desired. Even cycles are no danger for the model, as the only interesting thing is the *set* of access rights - and the *set* is not changed by going around more than one cycle. Cycles represent a set of roles having equal rights. Imagine three Webservers, where each Administrator role takes one of the other Administrator roles as member. In this example, Peter and Paul have got the Admin-rights all three WebServers.

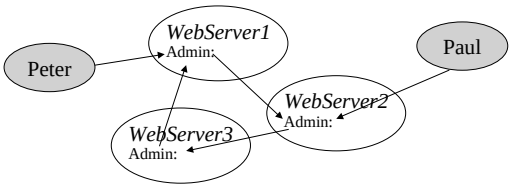

Figure 8 - Cycles of memberships

7.3.1 Extension of memberships

Static membership might not be sufficient for more complex environments. Also, with just adding members to a role, one cannot express mutually exclusive roles, which would need some kind of set-arithmetic.

We extend our model by the possibility that a membership definition can be either

- a fixed string - this is what we discussed so far
- a set expression - which can express requirements like "add all the members of role O1.x who are not member of role O2.y"
- an arbitrary program-like expression, which needs to be evaluated by a trusted[4] external interpreter and must return a set of members in the fixed string notation.

The set of members is evaluated at runtime – or better at query time. This makes a time dependent membership possible. Or memberships which depend on the fulfilment of interfaces etc.

The model does not care about the storage of the role members. The important fact is that a member can be added, removed and can be found in a role with defined methods. This can be assured using the interface idea again. But from our point of view this is much more an example of a abstract method which has to be implemented.

The results of the membership statements are united to make up the set of members of a role at this very moment in time.

7.4 Attributes

As the OO-paradigm implies, objects may also have attributes. Those attributes can be used for different purposes and can be again complex data structures. Attributes are used to store information about

- the users (in an object representing a user), i.e. name, address, age, staff number, etc.
- Email delivery: Email address and public key identifier
- GUI description: for the appearance in different GUI applications
- for applications to store state information about an entity (user/group/role, etc.)

The access control on attributes is managed over corresponding methods (getX and setX where X is the attribute name). Only initiators matching the condition in these methods have access rights to these attributes.

8 Conclusion and Future Work

The arrangement of possible targets of access into resource contexts, combined with a role-based differentiation of access modes of each resource provides a wide flexibility in modelling enterprise structures and relations (post hierarchy, groups) and makes the access control system manageable by non-computer-specialists. It also addresses the problem of controlling access to the access control information.

Most of this modelling approach is already implemented and used in some projects. We are still considering extending it in order to make it flexible enough to cover a wide area of applications. We are also moving our management interface from HTML-based to Java-based applets.

The revised model can be distributed, scaled and supports multilateral security. The TrustManager will be used for access control in the MultiPLECX context and is also planed to be used in administrative sectors at the Technical University of Berlin.

[4] The discussion of a trusted computing base (TCB) is out of scope in this paper. There will be another paper published talking about the deployment issues (operating systems, software packages, etc.)

9 Literature

[Nagel98] Jörg Bartholdt, Klaus Nagel, Thomas Hildmann, "Abgesicherte Internet-Umgebungen mit Hilfe rollenbasierter Zugriffsmechanismen für WWW- und Email-Dienste", Proceedings of "5. Workshop Sicherheit in vernetzten Systemen", DFN-CERT & PCA, p.1-36, March 1998 english version available at http://www.prz.tu-berlin.de/trustcenter/

[F5_E2S] E2S Consortium, „Deliverable F5: Trials/Demonstrator Summary", End-to-End-Security over the Internet, EU Project, TU Berlin, 1997, http://www.e2s.com/

[F1_E2S] Thomas Hildmann: „Deliverable F1", Secure Email Environment, End-to-End-Security over the Internet, EU Projekt, TU Berlin, 1997

[Thom97] Roshan Thomas, "Team-Based Access Control (TMAC)", Proceedings of the Second ACM Role-Based Access Control Workshop, p.13, 1997

[Grimm91] Grimm, Rüdiger, Sicherheit für offene Kommunikation. Teil 2: Kommunikation. Arbeitspapiere der GMD Nr. 598. Germany, 1991.

[Sand97] Ravi Sandhu et al.: „The ARBAC97 Model for Role-Based Administration of Roles: Preliminary Description and Outline", Proceedings Second ACM Workshop on Role-Based Access Control, Washington, Nov. 1997

[Guir97] Luigi Guiri & Pietro Iglio: „Role Templates for Content based Access Control", Proceedings Second ACM Workshop on Role-Based Access Control, Washington, Nov. 1997

[PM] Matt Blaze, Joan Feigenbaum, Jack Lay: "Decentralized Trust Management", DIMACS Technical Report 96-17, October 1996

[X.821] ITU Recommondation X.821: "Data Networks and Open System Communications Security: Open Systems Interconnection - Security Frameworks for Open Systems: Access Control Framework", November 1995

[Kuhn97] D. Richard Kuhn: "Mutal Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access Control Systems", Proceedings Second ACM Workshop on Role-Based Access Control, Washington, Nov. 1997

[Huang98] Yan-Huang, Ming-Chien Shan: "Policies in a Resource Manager of Workflow Systems: Modelling, Enforcement and Management", Hewlett Packard Technical Reports, HPL-98-156, September 1998